How initial task parallelism works

Apps using Tez have the ability to determine the number of tasks reading the initial external data for a job (the number of mappers in MapReduce parlance). Here is a short description of how that works.

- There is an InputInitializer specified for the initial vertex reading the external input. During vertex initialization, the InputInitializer is invoked to determine the number of shards of the external data distributed across the cluster. In MapReduce parlance, these would be called input splits and would be determined by the InputFormat for that external input.
- If Tez grouping is enabled for the splits, then a generic grouping logic is run on these splits to group them into larger splits. The idea is to strike a balance between how parallel the processing is and how much work is being done in each parallel process.
    - First, Tez tries to find out the resource availability in the cluster for these tasks. For that, YARN provides a headroom value (and in future other attributes may be used). Lets say this value is T.

      ```
      int totalResource = getContext().getTotalAvailableResource().getMemory();
      ```

    - Next, Tez divides T with the resource per task (say M) to find out how many tasks can run in parallel at one (ie in a single wave). W = T /M.
    - Next W is multiplied by a wave factor (from configuration - tez.grouping.split-waves) to determine the number of tasks to be used. Lets say this value is N.

      ```
      int taskResource = getContext().getVertexTaskResource().getMemory();
      float waves = conf.getFloat(
              TezSplitGrouper.TEZ_GROUPING_SPLIT_WAVES,
              TezSplitGrouper.TEZ_GROUPING_SPLIT_WAVES_DEFAULT);

      int numTasks = (int)((totalResource * waves)/taskResource);
      ```

    - If there are a total of X splits (input shards) and N tasks then this would group X/N splits per task. Tez then estimates the size of data per task based on the number of splits per task.
    - If this value is between tez.grouping.max-size & tez.grouping.min-size then N is accepted as the number of tasks. If not, then N is adjusted to bring the data per task in line with the max/min depending on which threshold was crossed.

      ```
      if (lengthPerGroup > maxLengthPerGroup) {
        // splits too big to work. Need to override with max size.
        int newDesiredNumSplits = (int)(totalLength/maxLengthPerGroup) + 1;
      ...
      } else if (lengthPerGroup < minLengthPerGroup) {
        // splits too small to work. Need to override with size.
        int newDesiredNumSplits = (int)(totalLength/minLengthPerGroup) + 1;
      ```

    - For experimental purposes tez.grouping.split-count can be set in configuration to specify the desired number of groups. If this config is specified then the above logic is ignored and Tez tries to group splits into the specified number of groups. This is best effort.

      ```
      int configNumSplits = conf.getInt(TEZ_GROUPING_SPLIT_COUNT, 0);
      if (configNumSplits > 0) {
        // always use config override if specified
        desiredNumSplits = configNumSplits;
      ```

    - After this, the grouping algorithm is executed. It groups splits by node locality, then rack locality, while respecting the group size limits.
- If the final number of splits is X then X tasks are created in the initial vertex and they are executed on the cluster to read the external data.

Here is the detailed explanation of grouping algorithm (TezSplitGrouper.getGroupedSplits).

1. Figure out desired number of grouped splits. This is affected by several factors. If TEZ_GROUPING_SPLIT_COUNT is set, the value will be used as initial count. Otherwise, the one passed in parameter will be used. Then the initial count will be corrected according to TEZ_GROUPING_SPLIT_MIN_SIZE and TEZ_GROUPING_SPLIT_MAX_SIZE: if the initial count causes a too small grouped split size, it will be overridden as total input size/TEZ_GROUPING_SPLIT_MIN_SIZE; if initial count causes too large grouped split size, it will be override as total input size/TEZ_GROUPING_SPLIT_MAX_SIZE.
2. Try grouping splits of same node to desired grouped split size. In this pass, we only allow splits in the same node to group together. Also, small grouped split is not allowed.
3. If we can no longer get a group on node level locality, there are two choices then. One is fallback to rack locality and continue to group, the other is to just allow small group on node level locality. The first one is by default, and the second one requires setting configuration TEZ_GROUPING_NODE_LOCAL_ONLY.
4. If there are still ungrouped splits, allow small groups anyway and continue to use locality level in last step.